武汉轻工大学

毕业设计（论文）

毕业设计(论文)题目： 数据库系统的探索与实践

姓　　名　郑小康

学　　号　**1505110164**

学　　院　数学与计算机学院

专　　业　软件工程

指导教师　蒋祖国

2019 年 5 月 15 日

摘要

在信息高速增长下， 数据始终处于举足轻重的地位,故而数据安全有效一直备受关注,各种数据解决方案层出不穷,而作为其基础支撑,数据库是必不可少。据调研目前在工业存在主流数据库有 Oracle、MySQL、PostgreSQL、TiDB、OceanBase。本文通过实际研发,叙述一个关系数据库(RDMS)的核心组件及实现过程。基于 JAVA 采用 LL 语法分析器对 SQL92 规范进行实现;数据刷盘采用的段区页的处理方式并按照固定页面格式进行存储;基于 JDBC 规范进行交互处理,旨在打造一个可实现单表 CRUD 的数据库。

关键词： 数据存储；语法分析；页面格式。

Abastract

Under the rapid growth of information, data has always played a pivotal role. Therefore, data security has always been a concern, and various data solutions emerge in an endless stream. As a basis for its support, databases are indispensable. According to the survey, there are currently mainstream databases in the industry: Oracle, MySQL, PostgreSQL, TiDB, and OceanBase. This paper describes the core components and implementation process of a relational database (RDMS) through actual research and development. Implementation of SQL92 Specification Based on JAVA Using LL Parser; the data processing method of the segment page used by the data brush disk and stored according to the fixed page format; based on the JDBC specification for interactive processing, Designed to create a database that implements single-table CRUD.

Keywords: data storage; syntax analysis; page format.

目录

1　引言

1.1 数据库系统的存在价值

无论是互联网产品,还是传统企业解决方案实现,最终都脱离不了数据,一切方便快捷及价值实现,而所有的数据最终都是通过某种特定规则进行存储,数据库就是这样一种基础设施组件。

数据库有效的实现了数据的持久化,可供系统在需要的时候及时进行调用,在不需要调用的时候又不必占用过多的资源。同时可以对数据操作进行相应的管理,实现特定角色只是具备相应特定权限,这样在一定程度能够保证数据的安全性。

1.2 数据交互操作的设计方案

数据交互核心是基于 JDBC 规范解析对应数据库资源进行操作,故在实现过程,对 JDBC 接口做了相应实现,在连接过程中,对相应的驱动进行加载。

由于是基于单库的增删改查,所以通过 URL 协议来判断是否创建数据库,并在连接过程,根据是否创建,将相应的数据库进行创建或者加载。在这个过程将上下文以及数据工厂和系统表等相应组件启动,为后面调用执行接口做准备。

在连接关闭之后,需要通过检查点机制,将缓存处理的数据脏页进行及时刷盘,以保证数据的一致性。

1.3 数据库系统设计概述

在设计过程中,首先设计就是连接处理过程,如 1.2,每个连接对应一个数据库的操作并配备相应的上下文。之后建立相应的系统表来存储用户表的基本信息,以供后面进行调度。SQL 语法主要是通过 LL 语法分析器进行分析,封装到对应句柄供分析器进行分析,从而生成相应执行结果集。数据持久化则是通过段分页方式,按照所制定的页面格式存储到磁盘。

1.4 基本实现目标

数据库系统的设计主要解决组件的解耦与联系,在本课题的系统下,核心组件功能如下:

（1） 系统之间的连接规范

（2） 系统功能组件之间的调度

（3） SQL 语言的语法分析

（4） 数据持久化的解决方案

（5） 数据表的基础信息维护

（6） 数据查询的优化方案

2　技术简介

2.1 SQL92 规范

SQL92 规范是数据库查询语言的第三个修订版本,在他之前具备 SQL86 和 SQL89 两种版本,之所以选择该规范进行执行,是因为它既足够稳定,又能兼容之前版本。在这里主要是根据其 SELECT、INSERT、UPDATE、DELETE 语句表达式进行实施处理。

2.2 JDBC 规范

JDBC 规范在首次被指定于 1997 年 1 月,在最初只提供基本的调用级接口,之后进行迭代,目前已经推送到了 JDBC4.2,支持一系列企业级功能,提供更为丰富的 API 接口,使得数据库资源的管理更为方便。

JDBC 规范的目标是提供 JAVA 语言数据库操作的解决方案,核心是维护 SQL,任何遵循与该规范的 JAVA 框架,都能通过 SQL 语句的方式操作数据库,故框架对 SQL 语句进行的封装或者映射必须基于 JDBC 所制定规则进行处理,这样则降低框架直接组合的复杂度,以及系统重构的成本。

2.3 LL(1)语法分析器

LL(1)属于一种自顶向下的分析方式。 自顶向下分析主要是通过递归的方式即采用 DFA 结合预测分析表生成相应的候选条件,进行执行,如下:

S->cAe

A->ab|a

根据最左推导原则,匹配 a 会选择 ab 产生式,这个时候匹配完毕 a 之后,e 是无法进行匹配的,故需要产生回溯,检测是否具有别的候选式可以进行选择。另外产生式还存在一个问题就是左递归,举例如下:

S->Sa|b

它的推导过程则是出现非终结符还是 S,这样 S 一直会循环,由于采用最左推导原则,出现左递归可能会导致程序一直处于循环状态,故所有的左递归处理方式都需要进行消除,具体方法可以有将其候选式改写为非直接左递归方式,举例如下:

S->bT’

T’->aT’

解决以上问题之后,在匹配字串的过程中,遇到一个文法存在多个候选式,还需要进行预测分析,选择对应的候选式, 在此时就需要生成预测分析表，结合预测分析表，构造当前字符下面能够匹配的所有终结符，在 LL 语法分析器采用的方式是，根据候选式体生成 FIRST 集,再依据 FIRST 集和候选式体构建 FOLLOW 集,做出对应的语句选择,就是 LL 语法分析器的主要过程了。

2.4 段区页分页方式及页面格式

数据分页采用的是段区页的管理方式,具体的数据落在对应的存储页的,存储页是分配信息则需要通过分配页进行管理,由于可能涉及到较多页面,所以引入了去的概念,一个字节八位,基于位图算法最多可以有 256 个有效位,故一个区能够存储 256 页,其中第一页是分配页,其余的都是存储页。一张表的数据由同一段进行管理,故将同一表的数据存储在对应段下的分区。具体信息则是存储到对应的页面,有页面对应格式判定当前页所属种类,具体过程见 5.2 数据刷盘。

2.5 B+Tree

B+Tree 是一种数据查询算法,是基于二叉树加操作系统固有特性逐步演变出来的一种数据库索引方案。由于操作系统具有局部性原理,一次预读的长度是页的整数倍,所以寻道时间会占用过高的成本,为了减少这一成本,就需要尽量减少磁盘 I/O 次数,而将一页存放尽可能多的数据和指针域刚好可以有效解决这一问题,于是出现了 BTree。B+Tree 则是 BTree 的一种变种,所有的非叶子节点存放的都是指针域,叶子节点则存放对应的数据。

2.6 动态生成字节码

基于 JAVA SE 规范,对已有抽象类字节码文件做补充,通过一系列字节码访问操作,生成相应的字段,方法和构造类的过程。

该技术在当前项目的使用场景:SQL 解析的语义节点具有不同种类,如解析的表,查询,插入等语义会存在不同的执行过程,以及语义之间可能存在组装,故需要基于抽象模板做具体实现。

2.7 平台开发环境

选项	说明
operating system	MAC
CPU	Intel i7
JDK	1.8.101
JVM	Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
RAM	16G
Version Control Tools	Git
编译器	IDEA

图 2-7 平台开发环境

3　系统分析

3.1 可行性分析

本文描述的是单表数据库增删改查的实现过程,具体存在核心技术有语义解析、数据刷盘、索引查找、数据字典、动态生成字节码等。

从经济可行性来讲,该系统设计明确,组件设计清晰,所以基于组件依赖性,设计相应开发流程,不断进行迭代演进即可。系统开发环境只需要在本机即可,待开发完成,单台机器部署即可,故不需要太大的经济成本。

从技术可行性来讲,由于本系统属于底层系统,不存在过程的技术依赖,大多数是基于理论进行开发实践。语义解析主要是基于 SQL92 规范,采用 LL 语法分析器,对 Token 通过有穷自动机的方式进行,最终调用对应的方法栈;数据刷盘则是基于段区页的方式管理数据;索引查找基于 B+Tree 的原则实现查询;数据字典则是相应的逻辑控制,存储用户表的数据关系,主要是逻辑实现;总而言之,各技术点目的明确,设计清晰,是可以实现的。

3.2 系统架构图

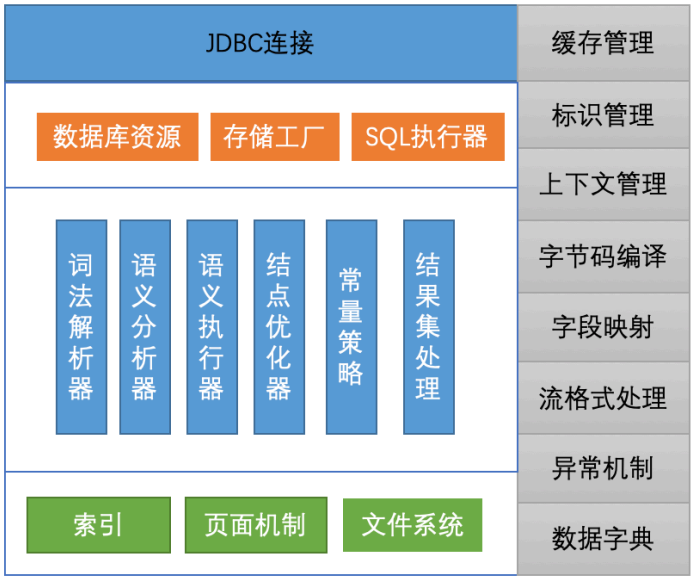


图 3-2 系统架构图

本系统设计的有客户端录入 SQL 语句,服务端解析对应的数据并进行持久化,故首先需要做的是建立连接,基于 JDBC 规范建立并获取对应的数据资源以及数据

工厂,之后将 SQL 语句交给 SQL 执行器进行执行,做相应的词法分析解析出对应的词法,之后根据语义解析,跳转到对应的执行方法做相应的封装,最后形成完整的语义并进行相应优化，比如对查询进行约束。在获取优化之后的节点后根据其生成对应的执行行为和结果集。至此完成全部编译,之后通过激活器进行激活,做对应的执行。在执行过程中，获取表对应的容器，根据容器找到对应的页，将数据以一定的格式存余页中。其中缓存管理、上下文、字典等作为基础组件支撑整个系统。

4 系统设计

4.1 设计目标

实现一个数据库系统,主要用于数据存储,主要是能对 SQL 语句解析并实现增删改查。

主要实现目标如下:

1.解析 SQL 语句生成对应的语义结点

2.针对语义结点生成对应结果集,构建对应的执行过程

3.通过 Btree 加快查询速度,快速找到指定的页

4.根据段分页的方式,将数据存储到磁盘

4.2 系统抽象设计

本系统主要是实现数据存储功能,从业务逻辑来进行抽象可以抽象出**连接层**、**语义层**、**数据交互层**、**中间设施层**。

连接层的主要作用是客户端远程连接服务，数据库服务启动后，打开相应的监听端口，远程客户端通过 TCP 连接,将对应报文按照 JDBC 规范进行解析,解析出对应的数据库资源和相应的数据工厂,进行加载。

语义层主要是进行词法分析,首先将一个 SQL 语句按照 DFA 法则分析出第一个词汇,跳转到对应的方法开始执行。针对每一个 Token,通过链式的方式执行下去,封装所有的信息,构建对应的语义结点。对语义节点做相应的优化以及加入到执行工厂进行处理。

数据交互层的主要作用是将数据存储到磁盘,为了有效利用磁盘空间,将磁盘按照段区页的方式进行管理,一个段对应一个磁盘文件,将一个文件划分为多个页,之后通过区的方式来管理这些数据页。

中间设施层的主要作用是做一些基础信息的维护,比如说数据字典和上下文管理等一些基础组件,做到整个系统能够正常运行,同时提供一些工具类将复用方法提取出来,减少软件开发复杂度。

4.3 系统总体架构设计

4.3.1 总体系统框架图

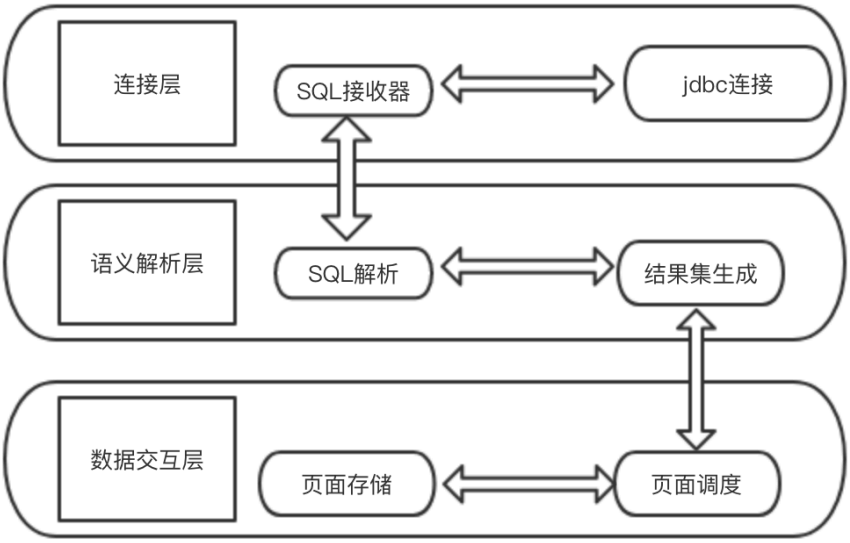


图 4-3-1 系统总体框架图

在系统中主要实现连接,再通过接收器接受对应的 SQL 语句,对语句进行相应的分析,分析出对应的结果集,以及构建回调结果集,在后面一系列执行按照相应的调度规则进行调度,基于常规段区页方式实现磁盘和数据之间的交互。

4.3.3 系统目录结构设计

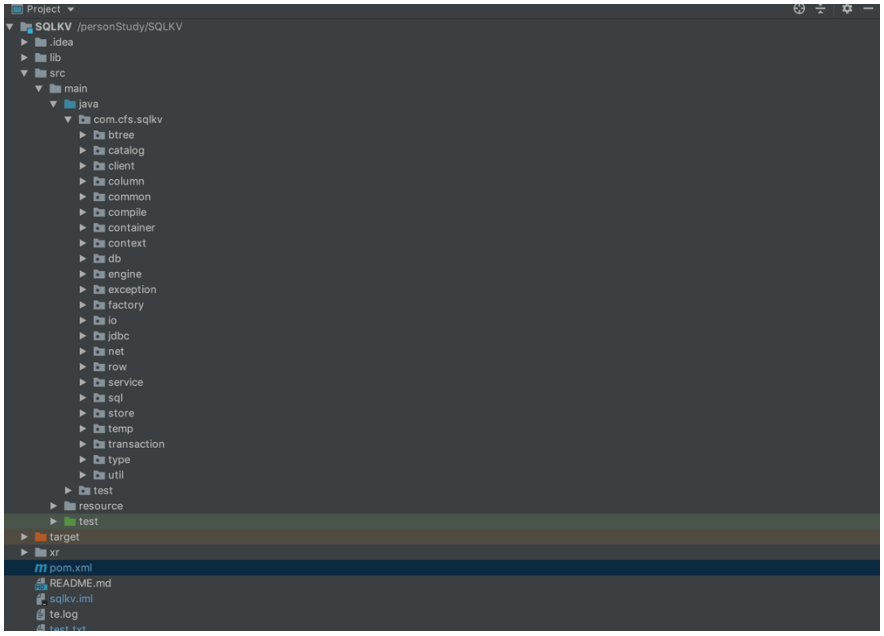


图 4-3-3 系统目录结构

项目管理使用的 Maven 的管理方式,基于约定大于配置原则进行实现,java 代码放在 java 对应文件目录,资源放在 resource 对应的文件目录下。统一包路径名 com.cfs.sqlkv 在其下面放置具体的功能模块。

5 系统实现

5.1 数据库连接设计

在这里以 JAVA 语言进行分析数据库连接，JAVA 语言采用的是 JDBC 规范(见 2.2 分析),在加载驱动过程,借助静态代码块的特性会进行一系列初始化,其中主要是将当前驱动给注册到驱动管理器,这是因为在通过驱动管理器进行连接的时候,会校验当前驱动是否注册,如果没有注册,则不允许进行建立连接,这样做的目的是确保驱动类存在;另一件事是在初始化过程,会将上下文服务创建。

建立连接的过程是从驱动管理器中获取所有有效的驱动,这些驱动都实现 JDBC 中的 Driver 接口,之后在连接的时候会调用模板方式 connect 的具体实现。对数据库进行初始化,其中包括数据库句柄,数据字典,上下文管理器等核心组件的创建与加载。

1. 数据库加载：

在进行连接建立的时候会创建一个数据库句柄,这个句柄在最开始是尚未与数据库资源建立关系的,在经过一系列的 URL 参数解析后,会获取数据库名称（dbname),以及是否创建数据库两个参数,之后将数据库句柄通过引用传递方式,传入到事务资源管理器进行处理加载。在事务管理器创建的过程中会根据 dbname 和持久化基础目录检测当前数据库是否存在,再结合 URL 解析的创建参数最后决定是否创建数据库,在数据库初始化过程会创建访问管理器、语言连接工厂、数据字典等组件。

2. 访问管理器：

访问管理器主要是决定创建 BTree 段还是常规段以及获取下一个创建的容器标识。容器标识的创建规则由容器类型和文件标号共同决定,数据工厂获取对应文件目录所有文件名称之后进行遍历找到最大的一个编号之后采用相应的自增策略即为新的文件编号,将新的文件标号左移四位之后与工厂类型进行或运算就是最后的容器标识,核心公式见下:

conglomid = (conglomid << 4) | factory_type

由公式可以看出,容器标识第四位代表当前容器类型,其余位则代表文件编号,同时二者共同组成具体的文件名,每一个容器对应一个文件,故文件编号采用的其实其自增的策略,每次加一,在文件上的体现是加 16。访问管理器本身还提供获取

工厂类型(BTree 工厂和普通工程),其就是根据 factory_type 来进行获取,至于具体的容器创建则是交给对应工程和缓存管理器去处理。

3. 语言连接工厂：

语言连接工厂顾名思义主要是做一个贯串连接的功能,当前实例提供获取语义、SQL 解析器、语言连接上下文、执行工厂等具体功能组件的接口。其中语义句柄对现有 SQL 文本进行封装,并且是作为后续语义解析的入口, SQL 解析器则是具体解析 SQL 的实例,语言连接工厂则是提供获取数据字典、语义上下文、事务管理器、激活器、数据值工程等一系列接口,执行工厂则是提供具体列的创建或者条件约束等。可以看出语言连接工厂句柄所提供的技术都是和 SQL 相关。

4. 数据字典：

数据字典主要存放基础核心表, 实现页面和表相关信息的调度管理, 具体有容器表、表信息表、字段列表、模式表, 具体见下；

表名称	标识号	描述
容器表	16	记录模式 Id 和表的标识以及容器 Id 等, 这样可以根据模式 Id 和表标识找到唯一的容器
表信息表	80	记录表标识, 表名, 表类型, 表模式等基础信息
字段列表	128	记录字段列名、字段列号、字段列数据类型等基础信息
模式表	176	记录模式 Id、模式名等基础信息

以上数据表会在数据字典创建或者加载过程中注入到相应的句柄, 并与相应的物理空间建立逻辑联系, 在表创建之后对应的索引页会进行加载。

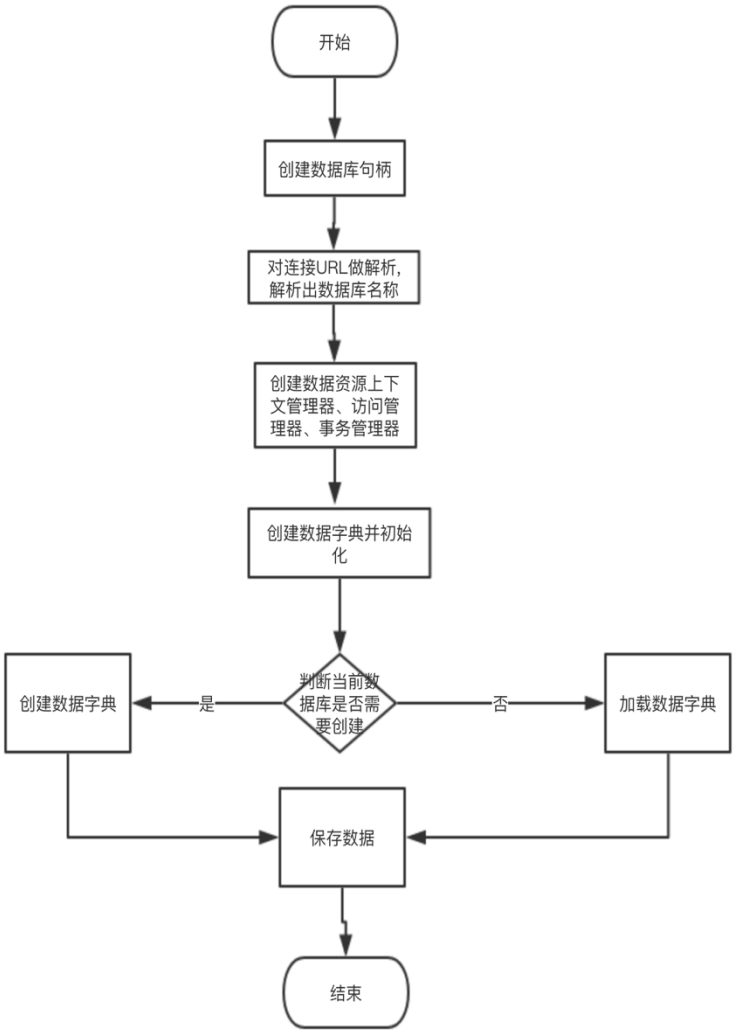


图 5-1 数据库连接简化逻辑

5.2 数据落盘

1. 缓存管理器：

数据最终目的就是持久化,这就需要保证数据最终有效的刷入磁盘,所有的数据基于缓存管理器进行实现,故需要设计一个缓存管理方案。见下图:

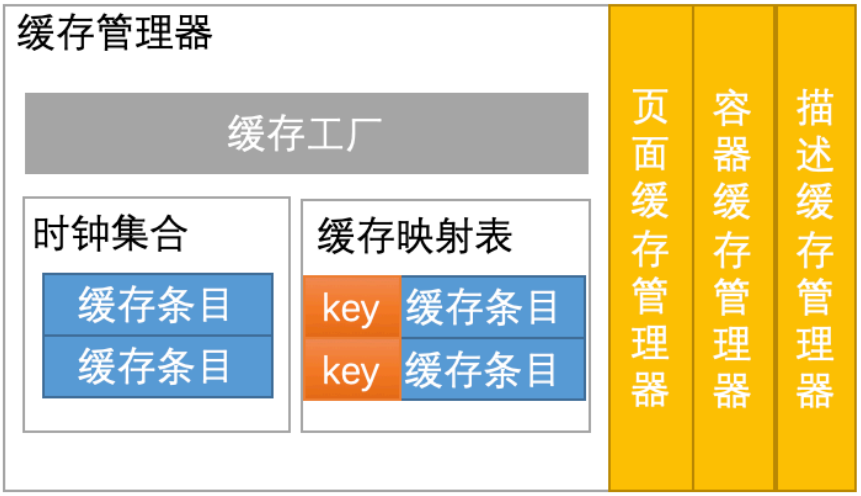


图 5-2-1 缓存设计方案

落盘数据存在几种类型,其中包括页面,容器,描述等类型,当然最终都是基于页面进行落盘。一个缓存管理器对应其中一种类型,缓存管理器本身是同一个类实现,只是不同缓存管理器里面的缓存工厂有所不同,这样在创建具体的对象时候,交给工厂去创建,故而实现了不同缓存管理器创建的工厂是不同的。至于缓存映射表的目的则是将缓存条目以 KV 映射的方式存储以方便下一次需要调用对象的时候可以直接通过 key 进行获取。时钟集合本身是一个集合对象,里面存储的缓存条目和缓存映射器里面是一一对应的,它的主要功能是对缓存条目进行管理。下面会对时钟集合和缓存工厂做单独描述:

在创建一个缓存条目的时候,会将其插入到对应的槽位即当前条目集合里面去,这个条目始终会带缓存对象句柄,这样就提供了缓存对象处理的能力。如缓存条目集合达到一定程度需要做清除处理,或者进行强制清除,都可以通过条目对具体的对象做清除操作。在清除缓存条目的时候会检查其缓存对象是否存在脏数据(脏数据指的是缓存中存在尚未刷新刷新到磁盘上的数据,在创建页或者修改页的时候会将 isDirty 进行修改来标识)。如果存在脏数据则需要将这些数据刷新到磁盘。缓存条目的清除是通过始终策略进行定期检查集合表,找到其中脏数据并进

行移除。

每一种缓存处理手段都和具体的缓存管理器相对应,见下表:

缓存管理器名称	缓存工厂
页面/容器缓存管理器	数据工厂
段缓存管理器	访问管理器
描述缓存管理器	数据字典

页面缓存管理器和容器缓存管理涉及页面数据处理,故其对缓存工厂对应的事数据工厂,在创建缓存对象的时候,会调用其 newCacheable 方法,方法签名如下:

public Cacheable newCacheable(CacheManager cm)

如果传入的缓存管理器是页面缓存管理器则会创建一个存储页句柄,反之则会创建容器句柄。段缓存管理器不涉及独立数据页,主要涉及到 BTree 和普通段,所以直接创建一个缓存段即可,至于是哪一种类型段在后面构建是根据 key 进行获取。描述缓存管理器主要是创建表描述,表描述会引用系统数据字典里面很多信息,所以直接使数据字典实现缓存工厂,这样在创建表描述的时候可以直接将数据字典作为句柄注入。

2. 页面处理逻辑:

在介绍页面分配、初始化、数据同步、清除功能之前应对页面结构做相应规划,具体规划见下图:

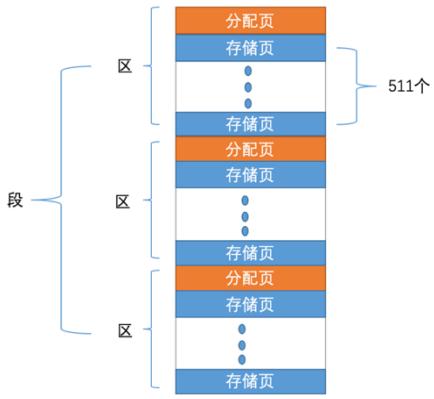


图 5-1 段区页的结构

一个段对应一个容器,普通表可以为一个段,一个索引键也可以为一个段,段表在磁盘上的体现是容器页,由容器页管理具体存储页。下面展示对应页的页面格式:

名字	长度	描述
FormatId	4	记录当前页的格式,决定当前页的类型
isOverflowPage	1	标识当前页是否是溢出页
pageStatus	1	记录页面的状态
pageVersion	8	记录当前页的版本
slotsInUse	2	一个槽位对应一个记录头,记录槽表正在使用的槽数
nextId	4	下一条记录 Id
deletedRowCount	2	当前页删除的记录数
freeSpace	N	空闲空间,当有了具体记录,这个空间会减少
槽位		每一个记录会对应一个槽位,槽位会记录记录的偏移量
CHECKSUN	8	校验和,用来保证当前页未被外部影响

5-2-2-1 存储页页面格式

存储页是用来存储记录,故包括基本信息和记录组织方式即可,对于记录组织是采用记录头的方式,从页面倒序开始,后八位是页面校验和,用来检验页面是否受到外部损害,之后是槽位, 每个槽位的格式如下:

每一个槽位的大小是固定, 记录偏移量占两个字节, 记录的长度占两个字节, 这样设置第一个槽位的偏移量位置, 公式如下:

slotTableOffsetToFirstEntry = (pageSize - CHECKSUM_SIZE - slotEntrySize);

基于这一计算规则,就可以确定每一个记录槽位所对应的位置,只需要基于首

个表槽位偏移量做减法即可。在找到对应的记录偏移量,对相应的记录进行处理,记录的格式如下:

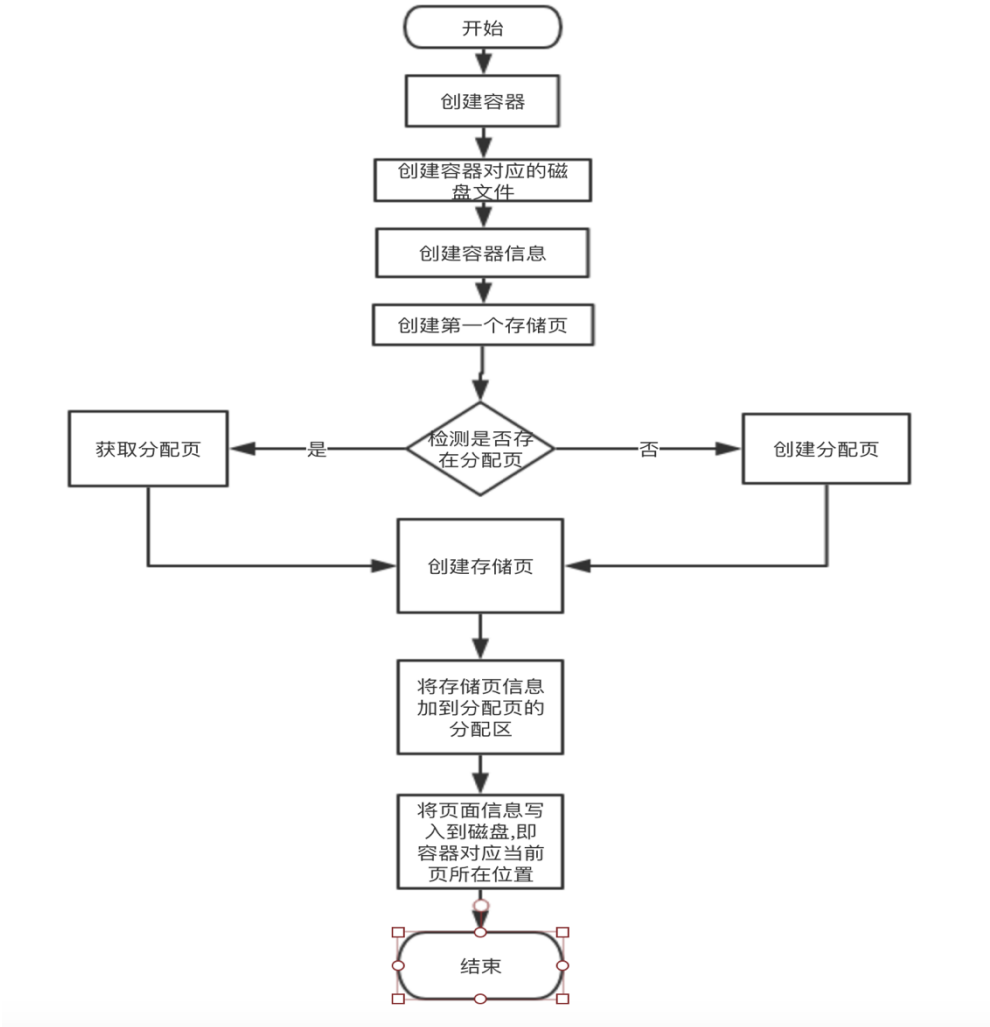
首先根据槽位提供的记录偏移量找到对应的记录,从开始获取固定格式的记录头,将信息封装到记录头对象并添加到页面内存槽位表可供下次获取,之后根据记录头可以采用遍历的方式对每一个字段进行处理,倘若部分字段不必处理可以采用 skipField 的方式,根据字段长度直接在遍历的时候跳过处理。

名字	长度	描述
FormatId	4	记录当前页的格式,决定当前页的类型
isOverflowPage	1	标识当前页是否是溢出页
pageStatus	1	记录页面的状态
pageVersion	8	记录当前页的版本
slotsInUse	2	一个槽位对应一个记录头,记录槽表正在使用的槽数
nextId	4	下一条记录 Id
deletedRowCount	2	当前页删除的记录数
nextAllocPageNumber	8	下一个分配页的页号
nextAllocPageOffset	8	下一个分配页的偏移量
BORROWED_SPACE_LEN	1	记录页面租借空间长度
borrowedSpace	N	租借空间具体内容,实际存储的是容器信息,在第一个分配页才存在
extentOffset	8	分区偏移量
extentStart	8	记录当前分区开始的页号

extentEnd	8	记录当前分区结束的页号
extentPageNums	4	记录当前分区页的数目
extentStatus	4	记录当前分区的状态

5-2-2-2 分配页页面格式

分配页是基于存储页做扩展,可以说它是特殊存储页,它存储容器信息,分配区信息以及其余存储页的信息,分配页处理逻辑如下:



5-2-2-3 分配页的实现过程

一个容器对应物理空间的一个磁盘文件,基于当前容器的分配页和存储页都是在这个容器对应的磁盘文件,只是一页的大小是 4kb,之后根据当前页面编号和大小

计算出页面的游标位置,将其存储到文件流对应位置。容器的创建在需要创建一张表或者一个 BTree 的时候进行创建,创建之后会进行后续操作创建第一个存储页,在创建存储页的时候需要找到分配页,在第一次的时候,首个分配页不合法,所以需要创建一个分配页,获取当前分配页之后会继续创建存储页,存储页创建完毕之后,会将存储页的页号等信息,记录到分配页的分配区,之后在将分配页的信息写入到磁盘文件时候,由于是容器进行操作,所以会检测当前分配页是都是第一个分配页,如果是,会将容器信息添加到分配页的租借空间,由此也可以看出一个分配页页面格式组成部分有存储页头部、分配页、容器和分配区。

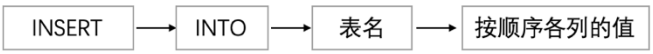
5.3 SQL 语句解析

SQL 语句的解析分为词法分析和语义解析两步,过程是先将语句转化为字符串流,从第一个字符开始,基于 LL 语法分析器生成的预测表,通过 DFA 找到第一个 ToKen,这个时候基于该 Token 进行消费跳转到对应的方法栈,找到对应的处理逻辑,在处理完毕之后,再基于字符串流分析上一个 Token 之后的游标,获取下一个 Token ,继续分析,直到将整个 SQL 分析完毕。下面基于 StatementPart 入口对基本 SQL 语句进行分析

1. SELECT 语句：

在对 SELECT 这个 Token 解析之后,下一步预测分析会出现查询的结果列,结果列存在通配和指定两种情况,如果是通配符则构建一个创建一个全列对象封装到结果列,否则基于逗号这个 Token 进行分割,每一个字符串作为一个列名,根据列名构建结果列,最终将这结果列象都添加到结果列集合中。在解析 FROM 以及后面的标识符获取表名,在解析 WHERE 以及后面的表达式,最终将所有信息封装到对应语义节点 SelectNode。

2. INSERT 语句:

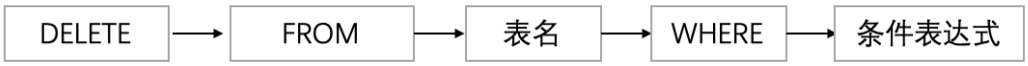


INSERT 语句的作用是将数据插入表对应的位置,在解析的时候首先需要消费 INSERT 和 INTO 两个 Token，之后解析标识符作为表名,再将所有的封装到结果列,将表名和结果列一起封装到 InsertNode。

3. UPDATE 语句:

UPDATE 语句是对现有的列进行修改,消费 UPDATE 之后,调用解析 UPDATE 语义的接口,首先解析标识符作为表名,封装相应的表对象,再解析 SET 这个 Token,将赋值表达式解析到列集合,目前只支持对单列的修改,再解析 WHERE 以及后面的条件表达式,最后将三者封装到 UpdateNode。

4. DELETE 语句:



DELETE 语句是对现有的列进行删除操作,主要就是将获取的表名和条件表达式封装到 DeleteNode。

以上四种语句操作情况都是按照正常情况解析,解析方式都是先获取下一个 Token,这个时候游标会回溯当前在获取下一个 Token 之前的位置,在具体调用匹配 Token 方法内部,才会进行实际的消费操作。在这个时候存在一种情况,就是预测分析表中不存在对应的 Token,那么会抛出解析异常。

5.4 语义结点生成结果集

所有根据 SQL 语义解析的结点最终都会通过父类 StatementNode 进行引用,采用模板方式都进行绑定生成相应结果集节点,由于大多数 SQL 语法都存在结果集节点,如 SELECT、Table、ResultCloumn、Row 等,故需要进行组织优化。在优化之后会对应优化的结果集节点进行动态生成相应的执行类去创建实际的结果集,供后续语义执行的时候,执行结果集从而实现数据记录的操作。对于不同语义结点具体实现方法是所有不同的,下面对 5.3 节 SQL 语义解析语义结点一一进行分析。

1. SelectNode:

查询结点最终基于游标进行查询,首先所做的事情就是绑定表,在 SQL 解析的时候解析了对应的表名,基于表名获取数据字典中的表描述,在获取表描述之后根据表描述获取表对应的结果集,以及获取相应容器描述信息为下面查询表数据提供支持。 以上表的基本信息加载完毕后,需要将对应的约束条件加载到表上面。再之后需要将显示的结果列给绑定。

在绑定完毕之后则需要做的是优化处理,针对于表生成基础结果集结点供后

续生成结果集,在这个过程需要将上文约束条件进行转移,以及将结果封装到基础结果集结点。

优化结点之后根据新的语义节点创建执行类,在这里是基于抽象类 BaseActivation 做字节码扩展,这是因为不同的节点一些基础支撑特性是固定的,不需要动态生成,则直接采用硬编码的方式即可。在获取到 BaseActivation 这个父类之后,将其作为句柄创建自动生成类,在生成构造器之后,生成创建结果集这一抽象方法的覆盖方法。获取容器信息,根据约束条件生成对应的不需要获取行的匹配字段。它的生成方式是根据条件生成 LocalField 添加到局部变量表,再结合关系操作调用执行工厂,将条件行为注入进去。在表结点的结果集生成之后,需要进一步对结果列的字节码进行生成,主要是将需要的结果列添加进去,最终生成相应的结果集类。采用递归的方式逐步包装,具体 SELECT->TABLE->ROW。

所生成的执行字节码文件封装到激活类中,后续通过激活类生成对应的对象进行执行。

结果集生成之后需要构建执行结点的常量行为,对于 SelectNode 这里是没有的。

2.InsertNode:

插入结点所涉及的是将数据给插入到表中,所以需要关心的是插入的值和表信息,在绑定的时候根据表名找到表描述,将解析出来的插入结果集和表描述所获取的容器一起绑定到插入结果集上。

3.UpdateNode:

修改结点需要考虑到对哪些匹配列进行修改,修改前值表达式和修改后值表达式以及表信息。在绑定的时候先绑定表信息,之后将约束条件绑定到表,所以说 InsertNode 是基于 SelectNode 结点实现,在完成 SelectNode 的绑定工作之后,在将修改的结果列绑定到插入结点。在生成字节码文件的时候根据装饰者模式的设计,生成查询对应的结果集,再将查询对应的结果集添加到插入结果集对应的句柄。

4.DeleteNode:

删除结点也是基于查询结点进行操作,在绑定的时候将查询结点对对应句柄注入到删除结点,在生成结果集字节码的时候同理将查询结果集注入到结果集里面。

5.5 结果集执行过程

结果集执行已经实际开始产生数据交互,对于不同操作结点,所执行的结果集句柄实例有所不同,但是都是以 open 为入口。

1. 查询：

查询结果集基于表扫描实现,故首先需要打开扫描控制器,之后获取一个模板行,将相应游标位置初始化。

基于迭代获取对应的行数据,在迭代中通过 next 等方法获取对应的行数据,先基于列描述信息构建相应的模板行,首先检测是否存在历史获取行数据,如果当前行游标位置小于以获取行数,则直接获取数组中的数据,这是因为每次获取的时候都会批量从页中获取部分连续行数据存入到 rowArray,如果游标位置大于 numRowsInArray 那么则再通过扫描控制器获取一组数据。

获取数据的过程,先通过让位置获取扫描页和对应的扫描槽位,之后基于槽位获取记录头,再根据匹配描述和记录头获取页面记录头对应的数据,若满足约束条件则将其添加到 rowArray。扫描完毕之后返回扫描获取的记录数。

客户端获取对应的列时候,只需要找到相应行即可以获取对应的字段数据结果。

2. 插入：

插入结果集则是需要将现有的数据给插入到页面,首先基于表获取可插入页,需要通过分区找到最后为装满的页。在获取到插入页之后,首先插入槽位标识并基于槽位标识和页面 id 创建记录标识通过页面行为将该条记录标识和行数据一起插入到页面,具体过程是先将其存入到一个临时缓冲流,再基于记录对应的偏移量拷贝到页面流对应的位置。

3. 修改：

修改记录首先根据容器和改变的列 id 创建对应行改变器，之后将容器给打开,根据查询结果集获取对应行,由于是修改,会构建对应的行位置,通过行修改器对行数据的修改添加到对应的位置。行修改器具体的修改过程,先根据改变列标识集合构建对应的改变行。

4. 删除：

删除记录行同样是基于行修改器实现，首先通过选择结果集打开容器扫描控制器，再根据容器 Id 和创建对应的行改变器并打开，之后获取对应的行。基于位置对对应位置进行删除，主要是将页面和内存中的槽位给无效化。

5.6 索引实现

考虑到时间成本,尚未实现 SQL 语义结点索引语法的实现,故索引的创建只存在于系统,在表创建过程会同时引导索引指定索引列进行创建。故在查询表信息的时候,可以检测到当前容器存在索引,故通过可以通过索引进行查询。

1. 索引创建：

在创建过程中，指定 implementation 为索引类型，则会获取索引容器创建工厂，创建对应的索引。通过访问控制器获取下一个可创建文件号，基于缓存管理器创建对应容器。

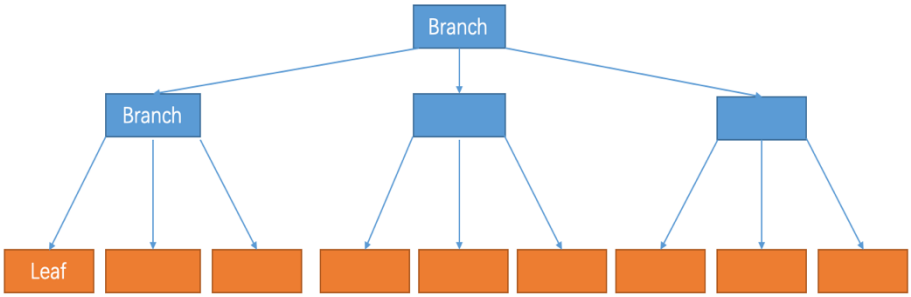


图 5-3-1 Btree 结构

初始化只会添加一个叶子，在 BTree 上除了底层为叶子节点，其它的就都是分支节点了，之所有没有标明根节点，是因为根节点在页裂变的时候可能会发生变化，具体在添加行的时候会进行说明。初始化会为当前 BTree 创建一页，并把 LeafControlRow 添加作为页面第一个槽位记录(BTree 的每一页都会对应一个节点，故每一也第一个槽位的记录都是控制行信息)，至此初始化完毕。

2. 索引查询：

在开始同普通表扫描一样，获取对应的索引页，构造查询参数，从根节点开始进行查询(当前控制性 Level 为 0 即为根节点会在初始化注入 root 句柄)，采用二分查找槽位对应的记录头进行匹配，最后将结果返回构造的查询参数。如果查询到则直接返回，如果没有查找到，根据槽位找到对应下一页，并将页面的控制行信息加载，进行递归查询，直到查找到叶子结点。

3.索引添加：

索引的添加主要是将每一条记录对应索引存入到对应索引容器,一个表存在几条索引,就会对应几个BTree,所以需要找到和BTrees对应的所有容器,逐一进行添加,

如果找到的容器页已经添加满了,同时到了叶子结点,则需要将这个叶子结点转化为分支节点,并做页分裂,创建新的页并初始化新的叶子节点控制行,之后将数据不断转入到叶子页。

结束语

无论是在互联网企业还是传统 IT 行业,应用的运行离不开数据,最终目的也是记录数据并可以进行有效反馈。

选择数据库一方面是这个数据库是一个很重要基础设施,另一方面是其计算机科学性很强,在实现本课题,对基础学科做了一个综合,在实现 SQL 语法的时候,采用了编译原理的 LL 语法分析器.在实现数据存储的时候采用了段分页存储方式,在实现查询索引采用了算法导论的 B+树和二分查找方式,并在本系统设计过程中有效结合设计模式实现以及对 JAVA 字节码的深入实践。虽然实现了基础增删改查逻辑,但仍存在很多不足,如事务解决方案和代码组织结构,还需要进行优化。

致谢

在毕业论文完成之时,由衷感谢数学与计算机学院各位恩师的关心和谆谆教导,无论是学习还是生活中遇到麻烦,老师们都给予很大帮助。在软件架构实验室里,有着我难以磨灭的回忆,感谢蒋祖国老师带我进入软件编程的大门,对我的启蒙之恩,没齿难忘。也谢谢那一群共同进步的同学好友们。软件工程主任贾瑜老师,不断引进校外优秀资源,对我们进行实训辅导,增长了我们的见识,同时贾老师那种对基础知识的严谨和谦和的风范也是我们数计学子的榜样。最终在蒋祖国老师的指导下做毕业设计,是对以前学习知识的检验,同时又是对综合能力的一次辅导,再一次由衷感谢老师指导和帮助!

参考文献

[1] (美) Alfred V. Aho。编译原理, 机械工业出版社, 2008。

[2] (美) Hector Garcia-Molina。数据库系统实现，机械工业出版社,2010

[3] (美) David R. O'Hallaron, Randal E. Bryant。Computer Systems:A Programmer's Perspective，

[4] (美) Erich Gamma。设计模式，机械工业出版社, 2000

[5] (美) Bruce Eckel。Java 编程思想，北京:机械工业出版社,2007

[6] (美) Tim Lindholm。Java 虚拟机规范，北京: 机械工业出版社, 2013.

[7] 姜承尧。MySQL 技术内幕，北京: 机械工业出版社,2013

[8] (美) Tapio Lahdenmak。数据库索引设计与优化，北京: 电子工业出版社,2015

[9] (美) Martin Fowler。重构: 改善既有代码的设计，北京: 人民邮电出版社, 2010

[10] (美) Jim Gray。事务处理: 概念与技术，北京: 机械工业出版社, 2004。